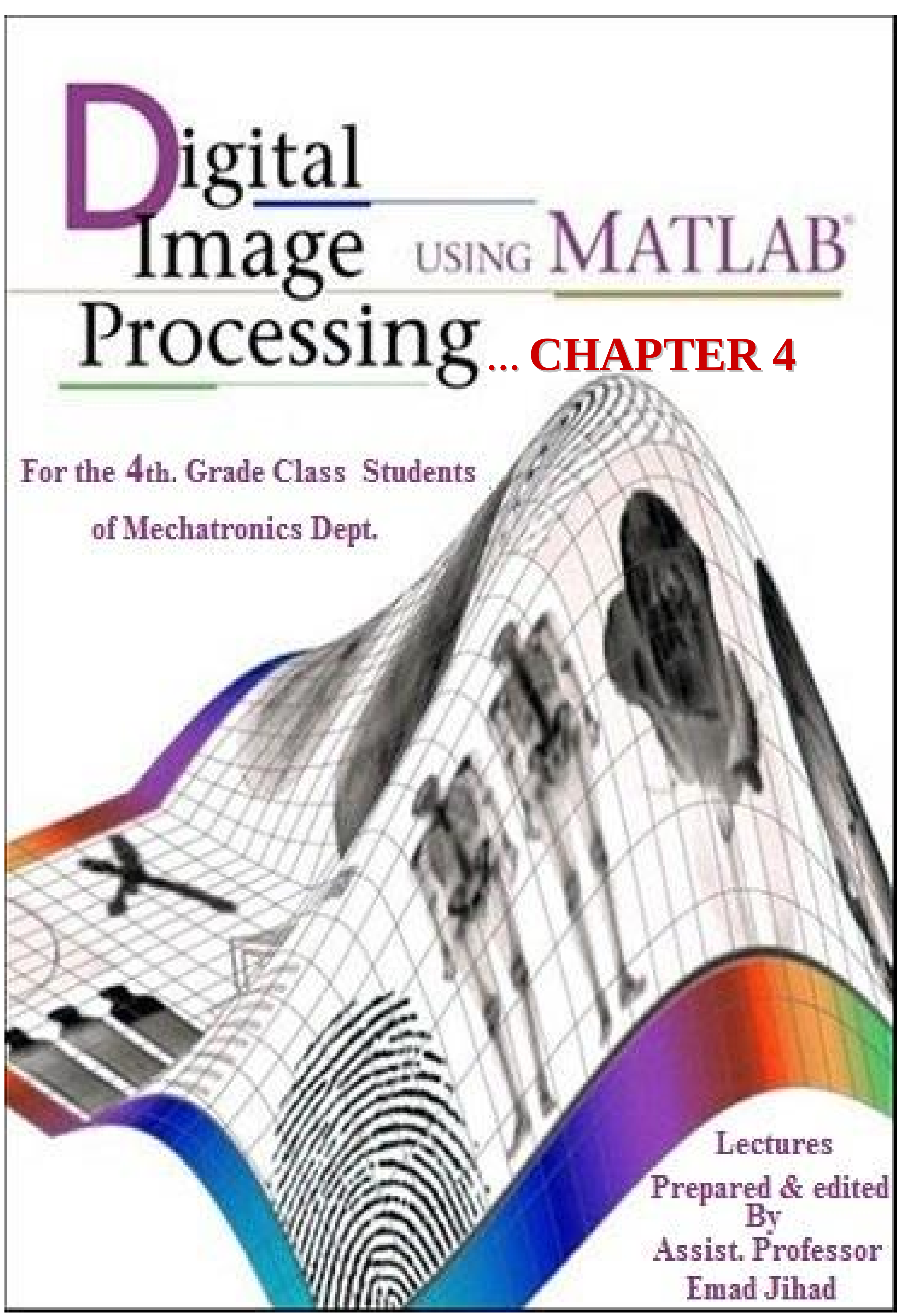

Digital
Image
Processing
USING MATLAB
... CHAPTER 4
For the 4th. Grade Class Students
of Mechatronics Dept.
Lectures
Prepared & edited
By
Assist. Professor
Emad Jihad

4- Geometric Image Transformations

4.1 Resizing an Image (Scaling)

There are several technique to change image size (Resizing or Scaling), and here we will begin with one that uses gray pixels losing technique to reduce (Shrink) image size, or increase it (Enlarge) by additional pixels.

a- In the following, we will focus on reducing a gray level image size by losing pixels in image matrix rows and columns according to reduction required.

- Let r = No. of rows, and c = No. of columns in the original image matrix.
- Use zero matrix (s) with r/2 and c/2 elements.
- Transfer original matrix element values to the new matrix (s) as illustrated in Fig (4-1) below:

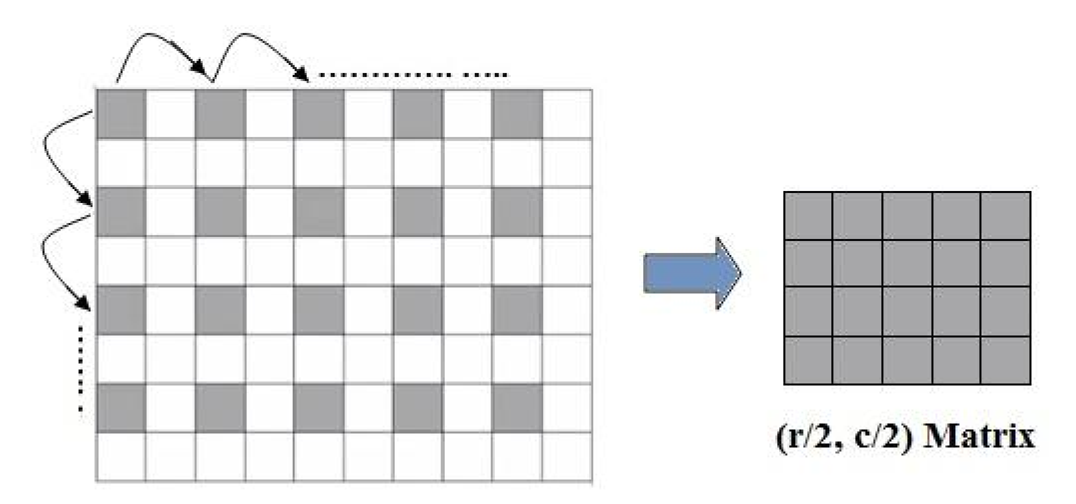


Fig (4-1)

The arrowed curves represent the (jump) required to reduce the number of pixels in both row-wise and column-wise, thus to get a r/2 and c/2 element that to be transferred to the new matrix (s).

b- Or, we can use Direct Matlab function which is much easier to perform all scaling upon any image type (except indexed, for now) using: **imresize,** the Syntax is:

```
B = imresize (A, scale)
B = imresize (A, [mrows ncols])
```

B = imresize (A, scale) returns image B that is scale times the size of A. The input image A can be a grayscale, RGB, or binary image. If scale is between 0 and 1.0, B is smaller than A. If scale is greater than 1.0, B is larger than A.

B = imresize (A, [mrows ncols]) returns image B that has the number of rows and columns specified by [mrows ncols]. Either NUMROWS or NUMCOLS may be NaN (Undefined number), in which case imresize computes the number of rows or columns automatically to preserve the image aspect ratio.

There are additional options in this function that will not be considered in this chapter. (For more details, see the Matlab Documentary or search help for imresize function!).

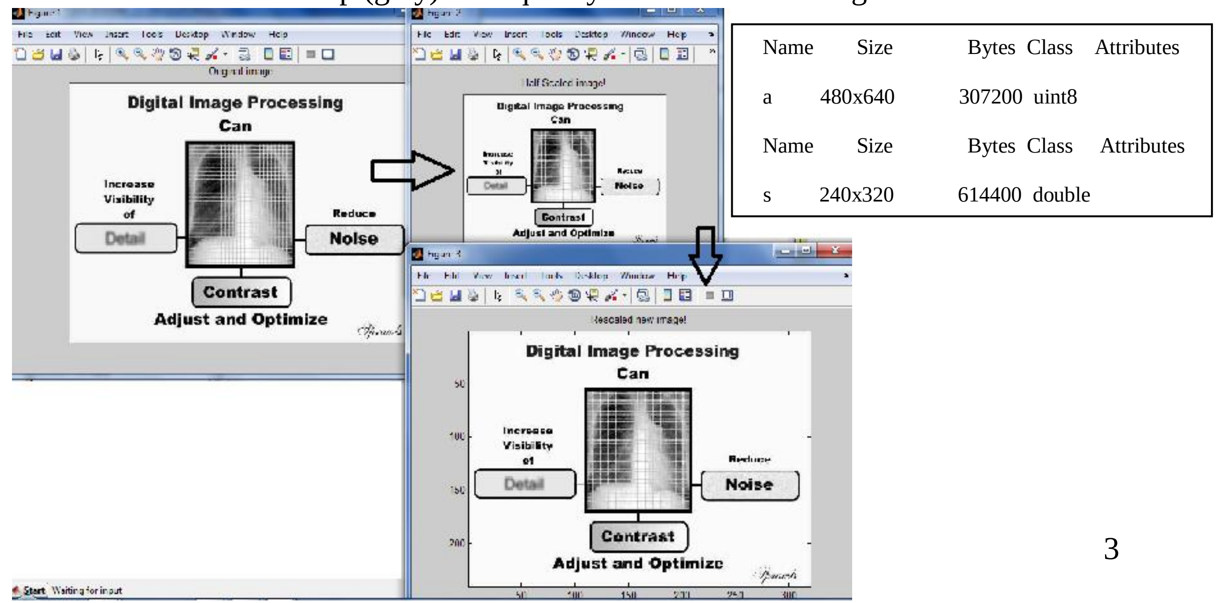
Example 4-1-1: Write Matlab script to perform image scaling ratio factor of 1/2 size of any input image using the traditional (hard way) method, this procedure is under user control for any number of images until decided to discontinue?

Solution:

```
%*************************************************************
%* Scaling Gray level images by matrices processing method!*
%* (Reducing an image size to half)!                       *
%* Program Name: ImageProEx13                              *
%*************************************************************

u = 'y';
while (u == 'Y') || (u == 'y')
    clc
    a = input ('Enter image file name: ', 's');
    a = imread (a);
    a = rgb2gray (a);
    size (a)
    [r,c] = size (a);
    i =1; j =1;
    s = zeros (r/2, c/2);
    for x = 1:2:r
       for y =1:2:c
           s(i,j) = a (x,y);
           j = j+1;
       end
    i = i+1;
    j =1;
    end
    imshow (a); title ('Original image');
    figure, imshow (s/255); title ('Half Scaled image!');
    figure, imagesc (s); colormap (gray); title ('Rescaled new image!');
    u = input ('Do you want to Continue (Y/N) ? ', 's');
    close all
end
close all
```

Notice that in order to show the new image matrix (s), we divide it by 255 because this new image is of the Class double (as mentioned in chapter 2), and that to rescale it we need to use colormap (gray). The quality of the rescaled image will be low!



Name	Size	Bytes	Class	Attributes
a	480x640	307200	uint8	

Name	Size	Bytes	Class	Attributes
s	240x320	614400	double	

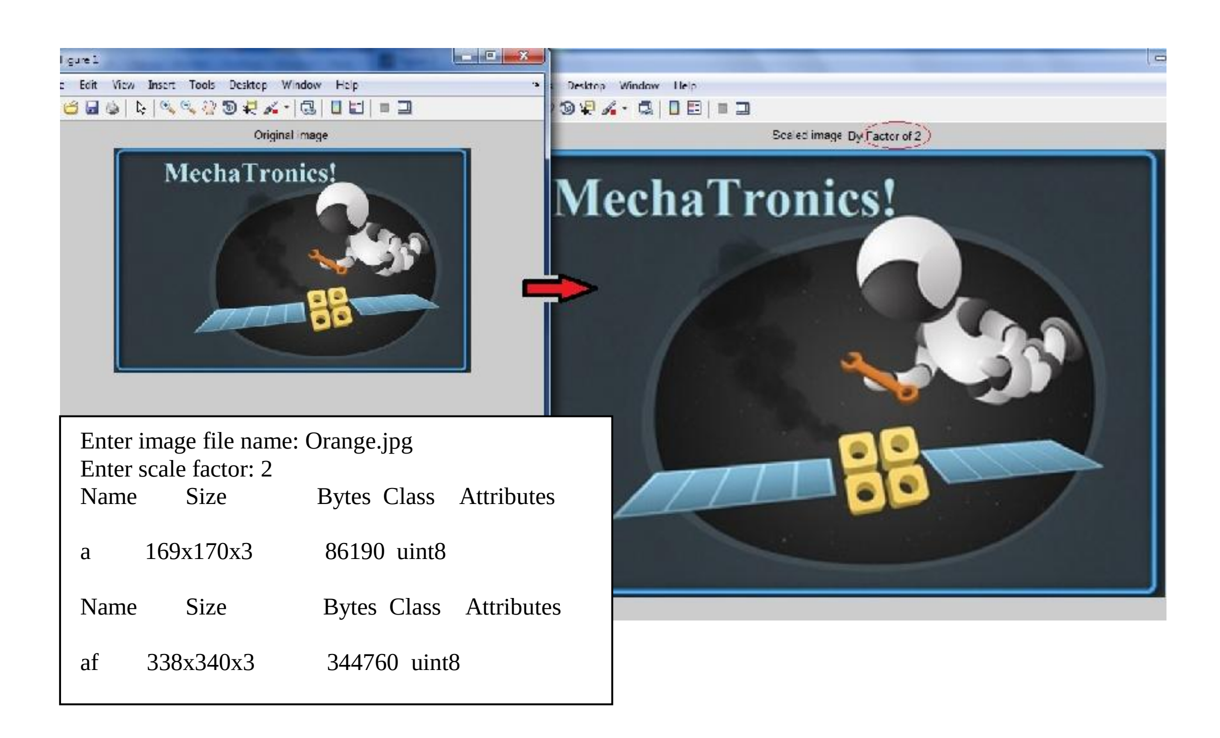
Example 4-1-2: Write Matlab script to perform arbitrary image scaling ratio (factor) of any type of images using Matlab's built in function. The user is given the choice control to end the program or continue?

Solution:

```
%*********************************************************
%* Image Resizing (Scaling) Using Matlab imresize function!*
%* Program name: ImageProEx24                             *
%*********************************************************
q = 'Y'
while (q == 'Y') || (q == 'y')
    clc
    a = input ('Enter image file name: ','s');
    a = imread (a);
    imshow (a);
    title ('Original image');
    f = input ('Enter scale factor: ');
    figure;
    af = imresize (a,f);
    imshow (af);
    title (['Scaled image by Factor of ',num2str(f)]);
    whos a, whos af
    q = input ('Do you want to continue (Y/N)? ','s');
    close all

end
```

The following is the input image and the output image after scaled by 2 times factor.

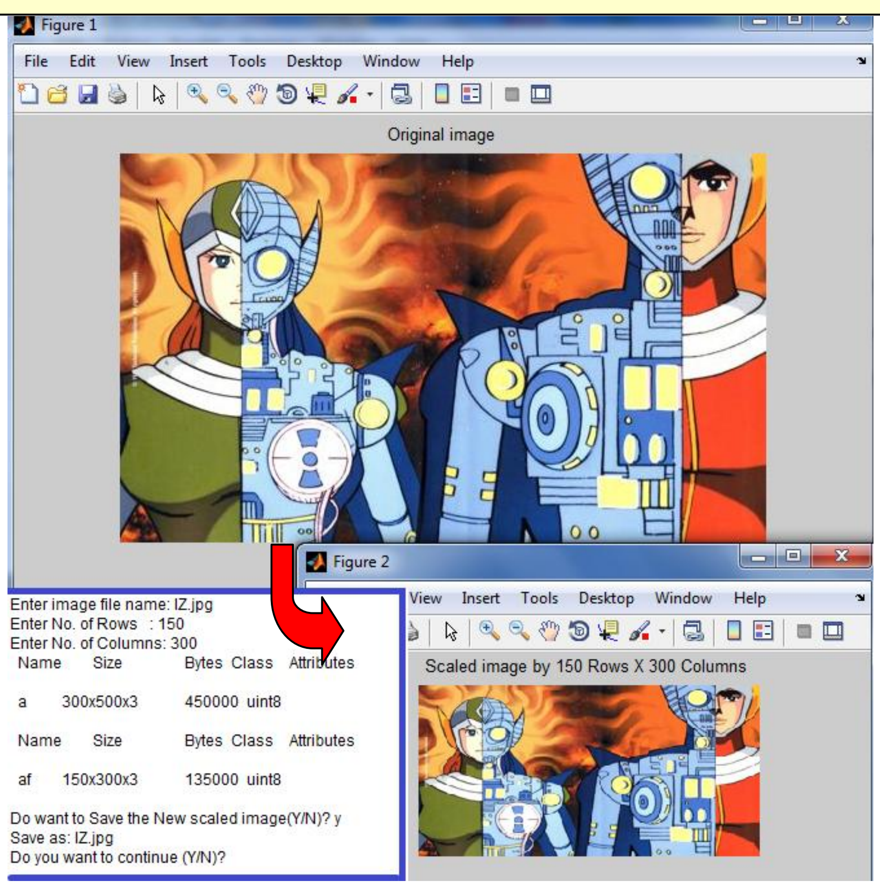


Notice that the Matlab's built in function (imresize) is much easier and flexible than the conventional method used in the previous example!

Example 4-1-3: Write Matlab script to perform arbitrary image scaling Rows and Columns of any type of images using Matlab's built in function. The user is given the choice control to save the new image and to end the program or continue?

```
%*************************************************************
%* Image Resizing (Scaling) Using Matlab imresize function!*
%* (Rows and Columns) option to be used for resizing.      *
%* Program name: ImageProEx24a                             *
%*************************************************************
q = 'Y'
while (q == 'Y') || (q == 'y')
    clc
    a = input ('Enter image file name: ','s');
    a = imread (a);
    imshow (a);
    title ('Original image');
    Rs = input ('Enter No. of Rows    : ');
    Cs = input ('Enter No. of Columns: ');
    figure;
    af = imresize (a,[Rs Cs]);
    imshow (af);
    title (['Scaled image by ',num2str(Rs),' Rows X ', num2str(Cs),' Columns']);
    whos a, whos af
    w = input ('Do want to Save the New scaled image(Y/N)? ','s');
    if (w == 'Y') || (w == 'y')
        fn = input ('Save as: ','s');
        imwrite (af,fn);
        else
    end
    q = input ('Do you want to continue (Y/N)? ','s');
    close all

end
```



There is a special function also **impyramid** which is used to Reduce or Expand images in Pyramid style. Its Syntax is: B = impyramid (A, direction)

Description

B = impyramid (A, direction) computes a Gaussian pyramid reduction or expansion of A by one level. Direction can be **'reduce'** or **'expand'**.

If A is m-by-n and direction is 'reduce', then the size of B is ceil (M/2)-by-ceil(N/2). If direction is 'expand', then the size of B is (2*M-1)-by-(2*N-1).

Reduction and expansion take place only in the first two dimensions. For example, if A is an (RGB) 100-by-100-by-3 and direction is 'reduce', then B is 50-by-50-by-3.

Note that impyramid uses the kernel specified below:

$$w = \left[\frac{1}{4} - \frac{a}{2}, \frac{1}{4}, a, \frac{1}{4}, \frac{1}{4} - \frac{a}{2}\right]$$, where $a = 0.375$.

The parameter *a* is chosen to be 0.375 so that the equivalent weighting function is close to a **Gaussian** shape and the weights can be readily applied using fixed-point arithmetic.

The following Fig (4-2) is an example of Gaussian shape.

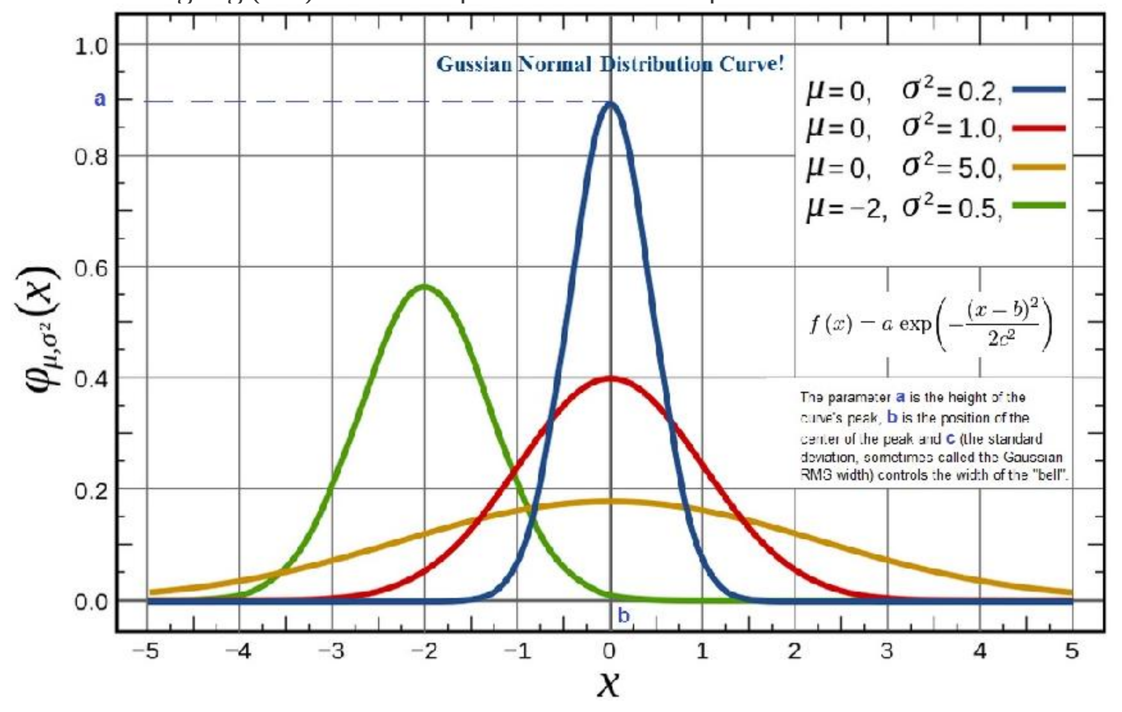


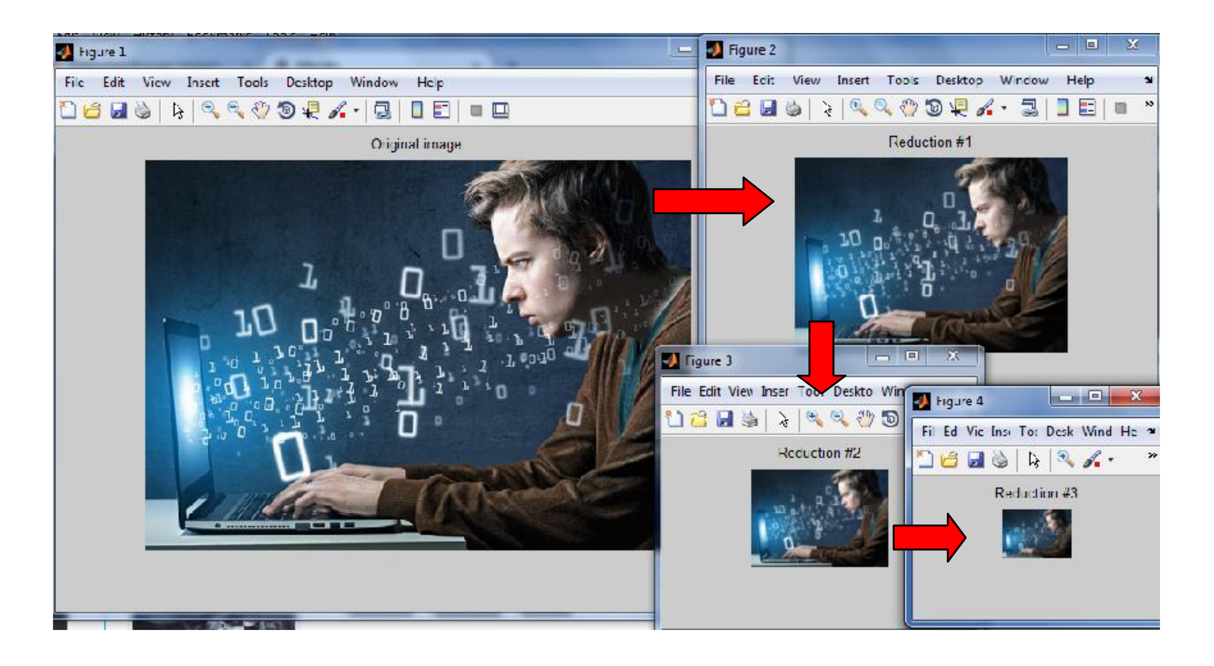
Fig (4-2): Gaussian distribution shape.

Example 4-1-4: Write Matlab script to perform image size Reduction and Expansion using 3 iterations for each of reduction and expansion processes?

```
%*************************************************************
%* Resizing images using Matlab built-in function(impyramid)*
%* Program Name: ImageProEx24b                              *
%*************************************************************
clc
I0 = input ('Enter image file name: ','s');
I0 = imread (I0);
imshow (I0), title ('Original image');
input ('Hit Enter key!');

for i = 1:3
        I0 = impyramid (I0, 'reduce');
        figure, imshow (I0)
        title (['Reduction #',num2str(i)]);
end
input ('Hit Enter key to Restore back!');
close all
imshow (I0), title ('Last reduced image');
for i = 1:3
       I0 = impyramid (I0, 'expand');
        figure, imshow (I0)
         title (['Expansion #',num2str(i)]);
end
input ('Hit Enter key to Exit!');
close all
```

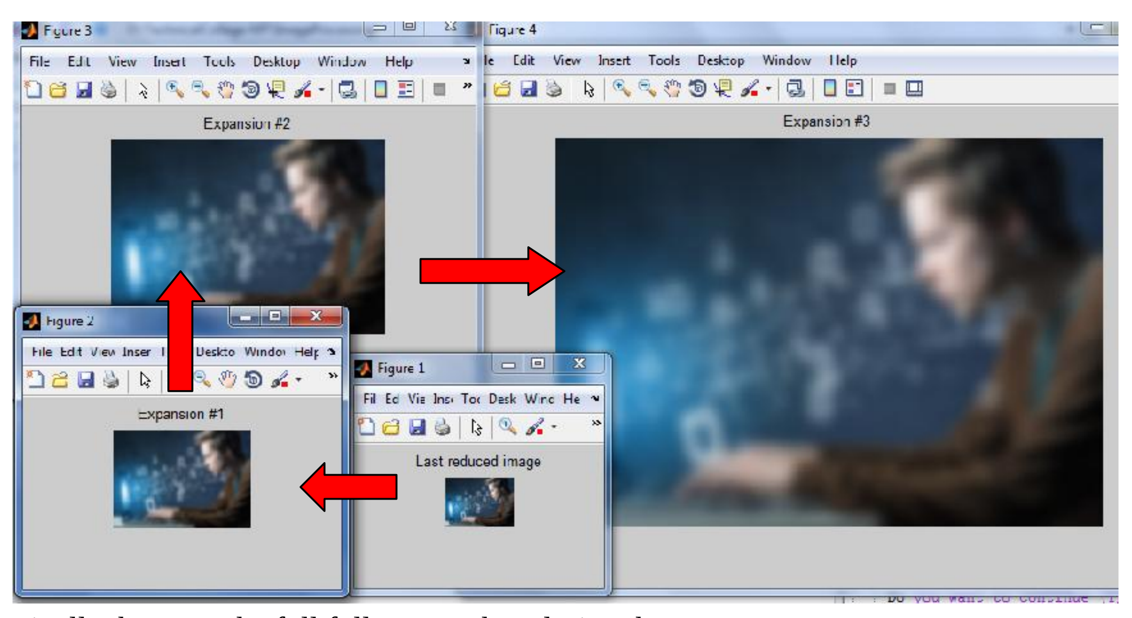
The Output figure images for reduction processes are as shown below:



Each reduction iteration process will shrink the number of pixels by (0.5), i.e. the number of rows will be **Rows/2**, and number of Columns will be **Columns/2** and so on…

Note: `input image matrix can be any numeric class except uint64 or int64, or it can be logical. The class of Output matrix is the same as the class of input matrix.`

While the Output of Expansion processes are as following… Notice the low quality (Blur) of this process which result from a reduced image size from previous reduce processes causing the lose of large number of pixels.



Finally, here are the full follow up taken during the program process:

```
Enter image file name: programmer.jpg
  Name      Size            Bytes     Class    Attributes

  I0        358x500x3        537000  uint8

Hit Enter key!
Iteration No.
     1

  Name      Size            Bytes     Class    Attributes

  I0        179x250x3        134250   uint8

Iteration No.
     2

  Name      Size           Bytes     Class    Attributes

  I0        90x125x3        33750     uint8

Iteration No.
     3

  Name      Size           Bytes    Class    Attributes

  I0        45x63x3         8505      uint8

Hit Enter key to Restore back!
Iteration No.
     1

  Name      Size           Bytes     Class    Attributes

  I0        89x125x3        33375     uint8

Iteration No.
     2

  Name      Size            Bytes     Class    Attributes

  I0        177x249x3       132219    uint8

Iteration No.
     3

  Name      Size            Bytes     Class    Attributes

  I0        353x497x3       526323  uint8
```

Image Size Reducing iterations

Image Size Expanding iterations

4.2 Rotating an Image

In Matlab, we can rotate any array elements (a vector or 2D matrix) counter clockwise using the function **rot90 (A, k),** thus a matrix (A) will be rotated counter clockwise 90 ^{o}, however it could be rotated to 180 ^{o} or even 270 ^{o} degrees if k is given the values of 2 or 3 respectively.

Example 4-2-1: Write Matlab script to rotate elements of the following arrays (A,B, and C) : 90, 180, and 270 degrees respectively counter clockwise:

A = [2 6 3 0 -5 1],

$$B = \begin{pmatrix} 7 \\ -2 \\ 6 \\ 8 \\ 1 \\ 3 \end{pmatrix}, \text{ and } C = \begin{pmatrix} 9 & 5 & 3 \\ 7 & 0 & 2 \\ 4 & -3 & 1 \end{pmatrix}$$

Solution:

```
%************ Rotating Matrix elements Counter Clockwise *************
clc
A = [2 6 3 0 -5 1]
B = [7; -2; 6; 8; 1; 3]
C = [9 5 3; 7 0 2; 4 -3 1]
disp ('Rotate A by 90, B by 180, and C by 270 deg. Counter clockwise!')
a = rot90 (A), b = rot90 (B,2), c = rot90 (C,3)
```

Results:
Rotate A by 90, B by 180, and C by 270 degrees Counter clockwise!
The new output matrices (a, b, and c) will have the new formation:

```
a =            b =            c =

    1              3              4     7     9
   -5              1             -3     0     5
    0              8              1     2     3
    3              6
    6             -2
    2              7
```

Thus, since images are just pixels in a form of matrices, the same is applied as in general matrices.

To Rotate a Gray level or Black and white image pixels (but not color), then its matrix elements will be rotated as well as done in general matrices case.

Theoretically, let's start with a counter clockwise rotating around the origin (0, 0). For a rotation over an angle ϕ a point (x, y) in the original image is mapped onto the point (x', y') in the resultant image. The relation between the points is:

$$x' = x \cos(\phi) - y \sin(\phi) \ldots\ldots\ldots\ldots (1)$$
$$y' = x \sin(\phi) + y \cos(\phi) \ldots\ldots\ldots\ldots (2)$$

In a geometric transform we need to express x and y as functions of x' and y'. To get from (x', y') to (x, y) we need to rotate over - ϕ. Show that we thus obtain:

$$x = x' \cos(\phi) + y' \sin(\phi) \ldots\ldots\ldots\ldots (3)$$
$$y = - x' \sin(\phi) + y' \cos(\phi) \ldots\ldots \quad (4)$$

The rotation algorithm then enumerates all points (x', y') in the resulting image, calculates the corresponding point (x,y) in the original image and uses an interpolation method to estimate the value of the original image in (x,y) and assigns that value to the point (x', y').

Matlab uses a special image powerful function **imrotate** to rotate any type of images (Clock-wise or Counter Clock-wise around its center). Syntax is:

B = imrotate (A,angle)
B = imrotate (A,angle,method)
B = imrotate (A,angle,method,bbox)

B = imrotate (A,angle) rotates image A by angle degrees in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for angle. imrotate makes the output image B large enough to contain the entire rotated image. imrotate uses nearest neighbor interpolation, setting the values of pixels in B that are outside the rotated image to 0 (zero).

B = imrotate(A,angle,method) rotates image A, using the interpolation method specified by method. method is a text string that can have one of these values. The default value is enclosed in braces ({}).

Value	Description
`{'nearest'}`	Nearest-neighbor interpolation
`'bilinear'`	Bilinear interpolation
`'bicubic'`	Bicubic interpolation **Note:** Bicubic interpolation can produce pixel values outside the original range.

B = imrotate(A,angle,method,bbox) rotates image A, where bbox specifies the size of the returned image. bbox is a text string that can have one of the following values. The default value is enclosed in braces ({}).

Value	Description
`'crop'`	Make output image `B` the same size as the input image `A`, cropping the rotated image to fit
`{'loose'}`	Make output image `B` large enough to contain the entire rotated image. `B` is generally larger than `A`.

(<u>Note</u>: Interpolation is the process used to estimate an image value at a location in between image pixels, it can be used in image resizing (scaling), Rotating …etc).

Example 4-2-2: Write Matlab script using Ordinary Matrix rotation method on a Color image with 90,180, and 270 degrees rotation.

Solution:

```
%***************************************************************
%* Rotating image matrix using Ordinary matrix rotation method*
%* For Gray and B/W images only!                               *
%* Program Name: ImagePro25                                    *
%***************************************************************
clc
A = input ('Enter image file Name: ','s');
K = imfinfo (A);  % Get image information
% Checking that the image is Not Color, otherwise converting it to gray
if K.BitDepth/8 == 3 disp('Color image is Converted to Gray level!');
   A = imread (A); A =rgb2gray(A);
else
end
figure
imshow (A);
title ('Original image');
figure
B = rot90 (A);
imshow (B);
title ('Image rotated 90 degrees');
figure
B = rot90 (A,2);
imshow (B);
title ('Image rotated 180 degrees');
figure
B = rot90 (A,3);
imshow (B);
title ('Image rotated 270 degrees');
input ('Hit Enter key to End');
close all
```

Example 4-2-3: Rotate any type of images by writing script by any arbitrary angle clock or anti clockwise, then use Nearest-neighbor, Bilinear and Bicubic interpolation methods to estimate image values at locations in between image pixels, Cropping the output image after using Bicubic interpolation method.

Solution:

```
%***************************************************************
%* Rotating image matrix using image rotation method          *
%* For any images type, with necessary interpolation method! *
%* Program Name: ImageProEx26                                 *
%***************************************************************
clc
disp ('Example about image Rotation techniques !');
disp ('=========================================');
ISource = input ('Enter image file name: ','s');
A = imread (ISource);
imshow (A), title ('Source image');
d = input ('Rotation Angle: ');
B = imrotate(A,d); figure;
imshow (B), title (['image Rotated by ',num2str(d),' Degrees']);
disp ('------------------------------------------');
disp ('Rotating image using: Nearest, Bilinear, and Bicubic int.');
B = imrotate(A,d,'nearest'); figure;
imshow (B), title ('Nearest interpolation method');
B = imrotate(A,d,'bilinear'); figure;
imshow (B), title ('Bilinear interpolation method');
B = imrotate(A,d,'bicubic'); figure;
imshow (B), title ('Bicubic interpolation method');
disp ('Crop method to Fit original image size!');
B = imrotate(A,d,'bicubic','crop'); figure;
imshow (B), title ('Using Crop to fit!');
disp ('==========================================');
input ('Hit Enter to End!');
close all
```

Notice that:

1- the Bilinear and Bicubic interpolation methods are much better than the default Nearest interpolation method. Images generated by using these methods are less aliasing (less jaggy) and thus could be useful as filters on rotated images.
2- Output rotated image size is larger than the source image size.
3- Also, if some one wants to keep the size of the generated output images as the same as the Source one, this may then Crop is to be used, but the rotated image probably will be clipped for the parts outside the original image size.

Hint: Use any pervious functions (such as size) to find out that the size of the image after interpolation methods, except the case of using Crop is preserved.

(Check the figures generated during the program in the next page).

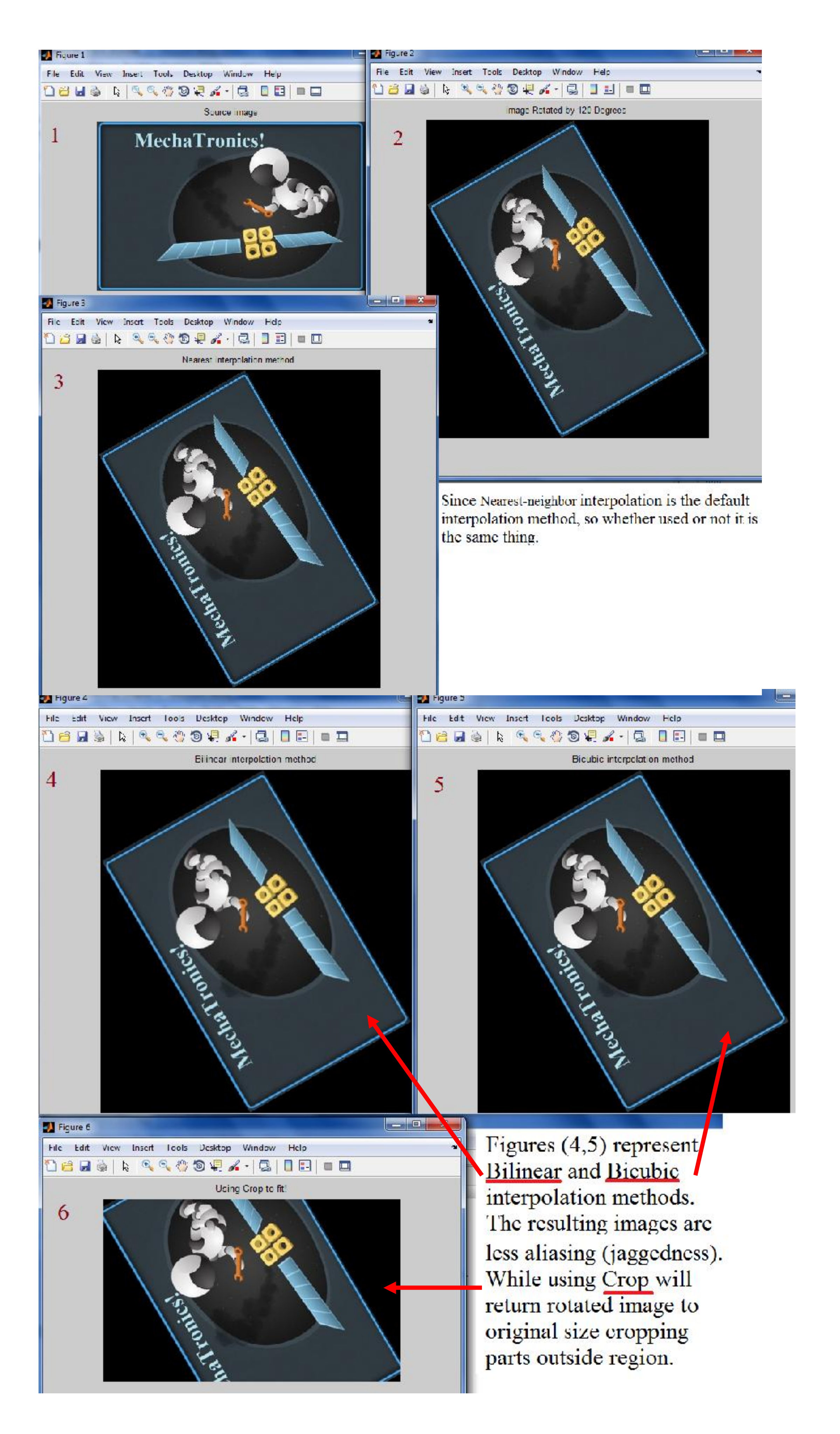


Figure 1
Source image
1
MechaTronics!
Figure 2
Image Rotated by 120 Degrees
2
MechaTronics!
Since Nearest-neighbor interpolation is the default interpolation method, so whether used or not it is the same thing.
Figure 3
Nearest interpolation method
3
MechaTronics!
Figure 4
Bilinear interpolation method
4
MechaTronics!
Figure 5
Bicubic interpolation method
5
MechaTronics!
Figure 6
Using Crop to fit
6
Figures (4,5) represent Bilinear and Bicubic interpolation methods. The resulting images are less aliasing (jaggedness). While using Crop will return rotated image to original size cropping parts outside region.

4.3 Cropping an Image

In Chapter 2, item 2-4, we gave an example of image crop using image tool, while in Chapter 3 item 3-1, we gave a script example to do crop too.

Matlab has a special function to perform this task directly using **imcrop** with syntax:

I2 = imcrop (I, rect)

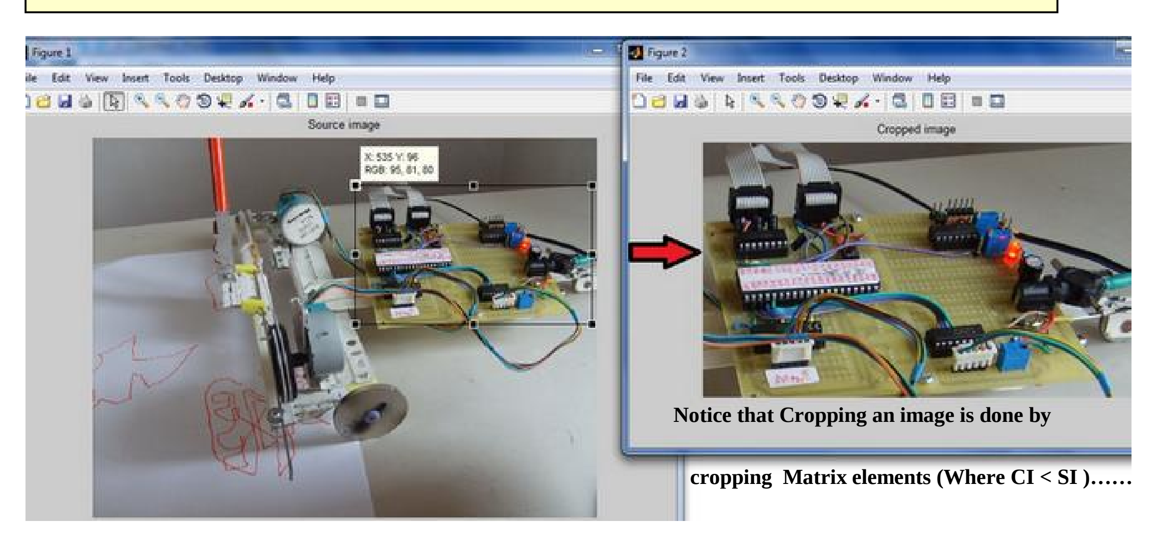
Where I is the input (source) image, I2 is the output Cropped image, rect is the rectangle frame information including: beginning pixel coordinates, Width and Height used to crop image.

Example 4-3-1: Apply image cropping on any given image through Matlab script?

```
%*****************************************************
%* Geometric Image Transformation- Cropping an image*
%* Program name: ImageProEx27                       *
%*****************************************************

clc; c = 'Y';
while (c == 'Y') || (c == 'y')

   A = input ('Enter image file name: ','s');
   SI = imread (A); imshow (SI); title ('Source image');
   disp ('----------------------------------------------');
   x = input ('Enter Beginning of Cropping frame for (x): ');
   y = input ('Enter Beginning of Cropping frame for (y): ');
   Width = input ('Enter Crop frame Width : ');
   Height= input ('Enter Crop frame Height: ');
   CI = imcrop (SI,[x  y Width Height]);
   figure, imshow (CI)
   title ('Cropped image');
   disp ('----------------------------------------------');
   c = input ('Try once more (Y/N)? ','s');
   close all
end
```



Notice that Cropping an image is done by cropping Matrix elements (Where CI < SI)……

4.4 Translating an Image (shifting)

A translation operation shifts an image by a specified number of pixels in either the *x* or *y* direction, or both.
In general, if a pixel of (x,y) is translated to new position (x',y') with values of Tx, Ty using the following equations:

$$x' = x + Tx \ldots\ldots\ldots\ldots(1)$$
$$y' = y + Ty \ldots\ldots\ldots\ldots(2)$$

Where Tx and Ty can take the integer values of (0 = > Tx > 0) , (0 > = Ty > 0).
This mean that if either Tx or Ty = 0 then there is No translation on that axis.
Positive Tx shifts the image to the Right, while goes to Down in case of Ty.
Negative Tx shifts the image to the Left, while goes Up in the case of -Ty.

The Translation process is written in Matrices form as following:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

… (Unified expression using matrices).

Since the translation of images is a little bit complex in Matlab, with no direct function to perform, then we will just be enough with these formulas.